UNIT-4
A8601

PRESENTED BY

M.YGANDHAR
Department of IT
Vardhaman College of
Engineering

01 TOPIC Collection classes-ArrayList

02 TOPIC LinkedList

03 TOPIC HashSet, and TreeSet

04 TOPIC EVENT HANDLING-Delegation Event Model

05 TOPIC Event Sources, Event Classes

06 TOPIC Event Listener Interfaces

07 TOPIC Handling Mouse and Keyboard Events

08 TOPIC Adapter classes

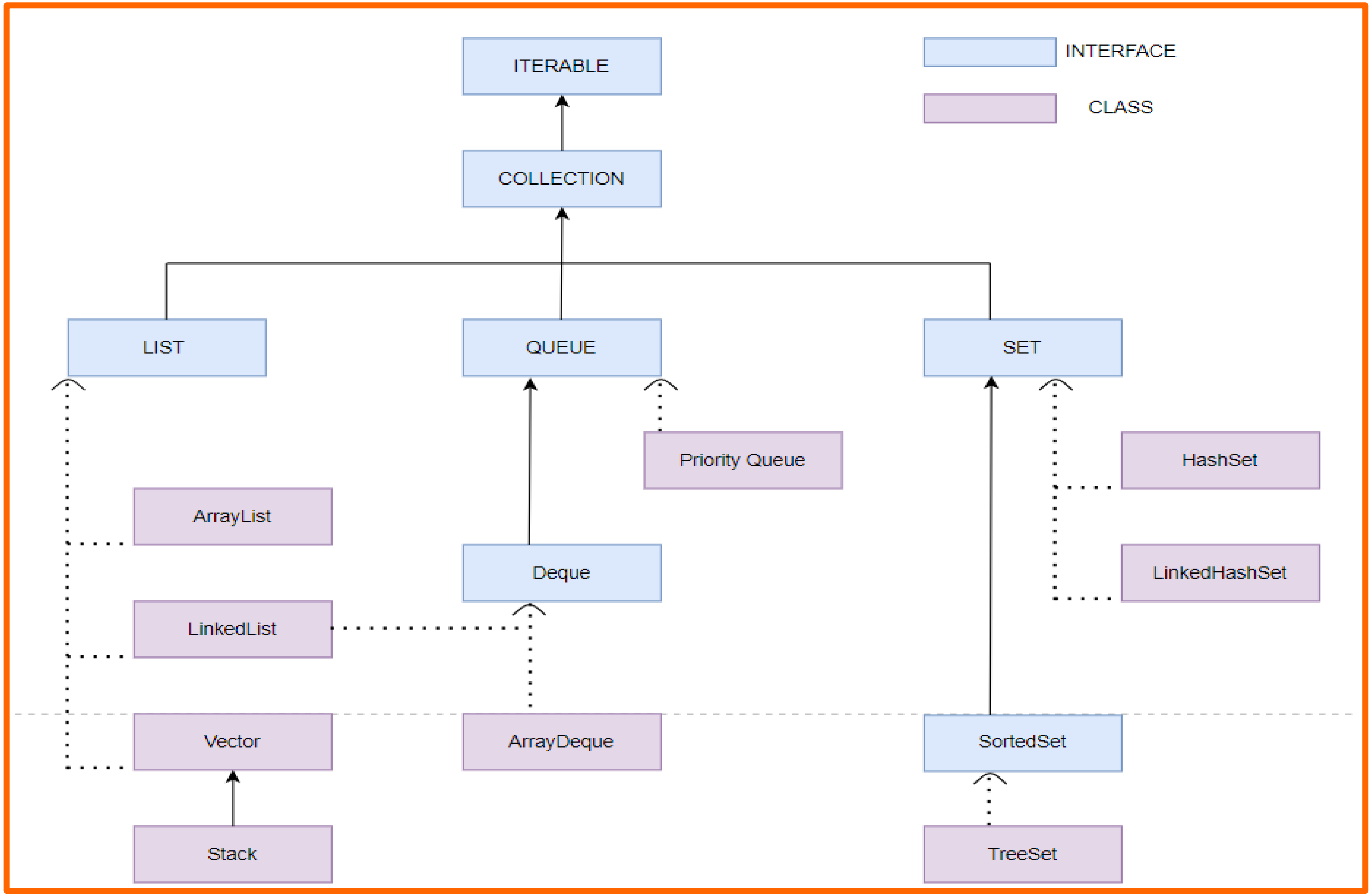
Collections Framework

- **Collections** in java is a **framework** that **provides an architecture** to **store** and **manipulate** the **group of objects.**
- The Java **collections framework provides** a **set of interfaces** and **classes** to implement **various data structures(**LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc)
- **All the operations** that you perform on a **data** such as **searching, sorting, insertion, manipulation, deletion** etc. can be **performed by Java Collections**.
- Java Collection simply means a **single unit of objects.**
- The **java.util package contains** all the **classes** and **interfaces** for **Collection framework.**

Example:

- The **LinkedList class** of the **collections framework** provides the **implementation** of the **doubly-linked list data structure.**

Hierarchy of Collection Framework



Hierarchy of Collection Framework

i.List Interface:

-The List interface **is an ordered collection** that **allows us to add** and **remove elements like** an **array.**

ii.Set Interface:

-The Set interface **allows us to store elements** in **different sets similar** to the **set in mathematics.**

-**Insertion order not preserved** i.e., They appear in the different order in which we inserted.

-**Duplicate elements** are **not allowed.**

-**Heterogeneous objects are allowed.**

iii.Queue Interface:

-The Queue interface is **used when we want to store** and **access elements in First In, First Out** manner.

Basic methods of Collection Framework

SNo	Method	Description
1	add(element)	It is **used to insert an element** in this collection.
2	addAll(collection_name)	It is used to **insert the specified collection elements in the invoking collection.**
3	remove(index/element)	It is used to **delete an element from the collection.**
4	removeAll(collection_name)	It is used to **delete all the elements** of the **specified collection** from the **invoking collection**.
6	int size()	It returns the **total number of elements** in the collection.
7	clear()	It **removes the total number of elements** from the collection.
8	contains(element)	It is **used to search an element.**
9	public Iterator iterator()	It **returns an iterator.**
11	boolean isEmpty()	It **checks if collection is empty.**
12	boolean equals(collection_name)	It **matches two collections.**

1. ArrayList

- **ArrayList** is a part of **collection framework** and it is **implements** the **List interface.**
- It is **present in java. util package.**
- It provides a **dynamic array for storing the element.**
- It is an **array** but **there is no size limit.**
- We can **add or remove elements easily.**
- It is **more flexible** than **a traditional array**.
- It can **dynamically increase** or **decrease in size.**
- Array lists are **created with an initial size.** When this **size is exceeded**, the collection is **automatically enlarged.**
- When an **ArrayList is created, its default capacity or size is 10** . The **size of the ArrayList grows** based on **load factor** and **current capacity.**
- The **Load Factor** is a **measure to decide when** to **increase its capacity**. The

1. ArrayList

- **ArrayList expands its capacity after each threshold** which is calculated as the product of **current capacity** and **load factor** of the ArrayList instance.

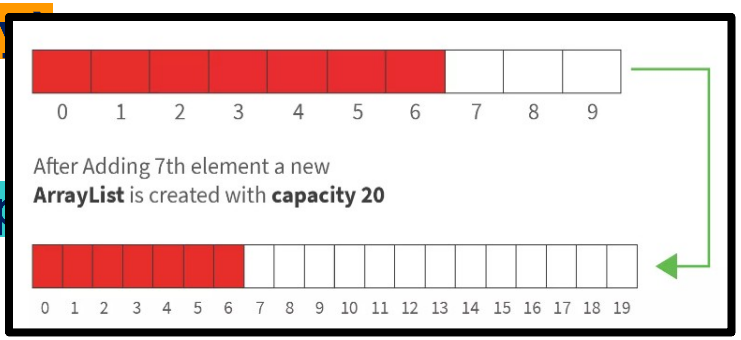
Threshold = (Load Factor) * (Current Capacity)

- For example, if the user creates an ArrayList of size 10,

Threshold = Load Factor * Current Cap

= 0.75 * 10

≅ 7



After Adding 7th element a new **ArrayList** is created with **capacity 20**

- This **means after adding the 7th element to the list**, the **size will increase** as **it has reached** the **threshold value.**
- **Internally, a new ArrayList** with **a new capacity is created** and the **elements present** in the **old ArrayList** are **copied in the new ArrayList.**
- The **new capacity** of the **ArrayList is calculated to be 50%** more than **its old capacity.**

new_capacity = old_capacity + (old_capacity >> 1)

- In the above formula, the **new capacity** is calculated as **50% more than** the **old capacity**.

How to create ArrayList

1.ArrayList<>(): It creates an **empty ArrayList** instance with **default initial capacity** i.e. 10

Syntax-1

```
ArrayList  <DataType>  VariableName = new
ArrayList <DataTYpe>()
```

Example

```
ArrayList  <int> a = new
ArrayList <int>()
```

2. ArrayList(int capacity): This constructor creates an empty ArrayList with initial
c an

Syntax-2

```
ArrayList <Datatype> VariableName = new
ArrayList <String> (size);
```

Example

```
ArrayList  <String> arr = new
ArrayList <String> (50);
```

Syntax-3

```
ArrayList <classname>  objname = new
ArrayList <classname> ( );
```

Example

```
ArrayList <Emp> obj = new
ArrayList <Emp>( );
```

i. ArrayList

```
// Demonstrate ArrayList.
import java.util.*;
class Demo
{
  public static void main(String[] args)
  {
    ArrayList< String> a = new
ArrayList< String>();
    a.add(“Java");
    a.add(“Python");
    a.add(“C-language");
    System.out.println(a);
  }
}
```

OUTPUT

```
[Java,Python,C-
language]
```

1. ArrayList Methods

SNO	Method	Example
1	add(element)	It is used to insert the specified element
2	addAll(collection_name)	It is used to **add all of the elements** in the **specified collection** to the **end of current list.**
3	remove(int index)	It is used to remove the element present at the specified position in the list.
4	removeAll(int index)	It removes all the elements from the list that are also present in the specified list.
5	get(int index)	It is used to find the index of particular element in the list.
6	set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
7	size()	It is used to find The length of the List .
8	clear()	It is used to clear entire list.
9	sort()	It is used to arrange entire list in ascending order.
10	reverseOrder()	It is used to arrange entire list in descending order.

1. ArrayList

```
import java.util.*;
public class TestA2
{
        public static void main(String[] args)throws Exception
        {
                ArrayList <String> a = new ArrayList <String>();
                ArrayList <String> b = new ArrayList <String>();
                a.add("Mango");
                a.add("Apple");
                a.add("Orange");
                b.add("Pineapple");
                b.add("Banana");
                b.add("Grapes");
                a.addAll(b);
                a.remove(4);
                System.out.println("Before removing all elements b is:"+b);
                b.removeAll(a);
                b.clear();
                System.out.println("After removing all elements b is:"+b);
                System.out.println(a);
                System.out.println("The size of the list b is:"+b.size());
                a.set(4,"Guava");
                System.out.println("After set list a is:"+a);
                System.out.println("Get the value from a:"+a.get(2));
                Collections.sort(a);
                System.out.println("After sorting list a is:"+a);
                Collections.sort(a, Collections.reverseOrder());
                System.out.println("After sorting list a is:"+a);


        }
}
```

OUTPUT

```
[Mango, Apple, Orange, Pineapple, Banana, Grapes]
After set list a is:[Mango, Apple, Orange, Pineapple, Guava,
Grapes]
Get the value from a:Orange
After sorting list a is:[Apple, Grapes, Guava, Mango, Orange,
Pineapple]
After sorting list a is:[Pineapple, Orange, Mango, Guava,
Grapes, Apple]
```

1. ArrayList

```
import java.util.*;
class Employee
{
        int eid;
        String ename;
        double sal;
        public Employee(int x, String y, double
z)
        {
                eid=x;
                ename=y;
                sal = z;
        }
}
```

```
public class EmpAlist
{
        public static void main(String[] args)
         {
                ArrayList<Employee>  list =new ArrayList<Employee>();
                Employee e1=new Employee(101,"Amar",75000.50);
                Employee e2=new Employee(102,"Akhil",85000.50);
                Employee e3=new Employee(103,"Anush",19500.50);
                list.add(e1);
                list.add(e2);
                list.add(e3);
                System.out.println("\n The number of employees are:" +list.size());
                System.out.println("\n The employess data is \n");
                for(Employee e:list)
                {
                        System.out.println(e.eid+":"+e.ename+":"+e.sal);
                        System.out.println();
         }
                list.remove(2);
                System.out.println("\n After removing number of employees are:" +
list.size());

       }

}
```

OUTPUT

```
 The number of employees are:3

 The employess data is

101:Amar:75000.5

102:Akhil:85000.5

103:Anush:19500.5


 After removing number of employees are:2
```

LinkedList class

- **LinkedList class** uses a **doubly LinkedList to store element**. i.e., the **user can add data at the first position as well as** the **last position.**
- If we need to perform **insertion /Deletion operation** the **LinkedList is preferred.**
- **LinkedList is used** to **implement Stacks** and **Queues.**

How to create a LinkedList

Syntax

```
LinkedList<DataType> VariableName = new
LinkedLits < DataType>();
```

LinkedList

```
//Program to Demonstrate LinkedList
import java.util.*;
class Test
{
  public static void main(String[] args)
 {
    LinkedList<String> cars = new
LinkedList<String>();
    cars.add("BMW");
    cars.add("FORD");
    cars.add("KIA");
    System.out.println(cars);
  }
}
```

OUTPUT

```
[BMW, FORD, KIA]
```

Methods of LinkedList

SNO	Method	Description
1	add(eelement)	It is used to **add the specified** element to the **end of a list.**
2	add(int position, element)	It is used to **insert the specified element** at the **specified position** in a list.
3	addAll(collection_Name)	It is used to **add all of the elements** in the **specified collection** to the end of this list.
4	addFirst(element)	It is used to insert the given element at the beginning of a list.
5	addLast(element)	It is used to **append the given element** to the **end of a list.**
6	getFirst()	It is used to **return the first element** in a list.
7	getLast()	It is used to **return the last element** in a list.

Methods of LinkedList

SNO	Method	Description
8	removeFirst()	It **removes** and returns the **first element from a list.**
9	removeLast()	It **removes** and returns the **last element from** a list.
10	removeFirstOccurrence(Object)	It is used to **remove the first occurrence** of the **specified element** in a list
11	removeLastOccurrence(Object)	It **removes** the **last occurrence** of the **specified element** in a list
12	lastIndexOf(Object)	It is used to return the **position** in a **list of the last occurrence** of the specified element, or -1 if the list does not contain any element
13	indexOf(Object)	It is used to **return the position** in a **list of the first occurrence** of the specified element, or -1 if the list does not contain any element.
14	get(position)	It is used to **return the element** at the **specified position** in a list.

LinkedList

```
import java.util.LinkedList;
class TestLL1
{
	public static void main(String[] args) throws Exception
	{
		LinkedList <Integer> a = new LinkedList
<Integer>();
		a.add(10);
		a.add(20);
		a.add(30);
		System.out.println(a);
		System.out.println(a.getFirst());
		System.out.println(a.getLast());
		a.addFirst(40);
		a.addLast(20);
		a.add(4,35);
		System.out.println(a);
		System.out.println(a.indexOf(20));
		a.removeFirstOccurrence(20);
		System.out.println(a);
		a.removeLastOccurrence(20);
		System.out.println(a);
	}
}
```

OUTPUT

```
[10, 20, 30]
10
30
[40, 10, 20, 30, 35,
20]
2
[40, 10, 30, 35, 20]
[40, 10, 30, 35]
```

Difference between ArrayList and Linked List

- The **ArrayList class creates** the list which is **internally stored** in a **dynamic array** that **grows** or **shrinks** in **size as the elements are added** or **deleted from it.**
- **LinkedList** also **creates** the **list** which is **internally stored** in a **DoublyLinked List.**
- **Both the classes** are **used to store** the **elements** in the **list.**
- **ArrayList allows random access** to the **elements** in the list as it operates on **an index-based data structure.**
- **LinkedList does not allow random access** as it **does not have indexes** to access elements directly, it has to traverse the list to retrieve or access an element

3. HashSet class

- HashSet **stores the elements** by using **Hashing mechanism.**
- It **contains unique elements** only.
- It **allows null values**.
- It **does not maintain insertion order**. It **inserted elements based on their hashcode.**
- HashSet is the **best approach for the search operation.**
- In HashSet **get() and set() method not present** because for **get and set method index is required** and in HashSet **elements stores** at a **random address**
- There are **three different way**s to **create HashSet:**

i.HashSet hs = new Hashset();

- ✓ Here, **HashSet default capacity** to store elements is **16** with a **default load factor/fill ratio of 0.75.**
- ✓ **Load factor** is if **HashSet stores 75% element** then **it creates a new HashSet** with **increased capacity.**

ii. Hashset hs = new Hashset(100);

HashSet

```
//Use HashSet methods to perform operations on
collection of data
import java.util.HashSet;
public class Test
{
    public static void main(String[] args)
    {
        HashSet h = new HashSet();
        h.add(7);
        h.add("A");
        h.add(4);
        h.add(3);
        h.add("Hai");
        h.add(null);
        System.out.println(h);
        System.out.println(h.add(4));
    }
}
```

OUTPUT

```
[null, A, Hai, 3,
4, 7]
false
```

HashSet

```
//Write a program to remove duplicate elements.
import java.util.*;
public class Main
{
	public static void main(String[] args)
	{
		int a[]={1,1,1,2,3,5,5,5,6,6,9,9,9,9};
		HashSet <Integer>  hs = new
HashSet<Integer>();
		for(int i=0;i<a.length;i++)
		{
			hs.add(a[i]);
		}
		for(int i:hs)
		{
			System.out.print(i+" ");
		}
	}
}
```

OUTPUT

```
1 2 3 5 6 9
```

HashSet

```
//Write a program to check 1 to 10 numbers are existing in hashset or not
import java.util.*;
public class TestHS3
{
    public static void main(String[] args)
    {
        HashSet<Integer> h = new HashSet<Integer>();
        h.add(8);
        h.add(3);
        h.add(7);
        for(int i = 1; i <= 10; i++)
        {
            if(h.contains(i))
            {
                System.out.println(i + " was found in the set.");
            }
            else
            {
                System.out.println(i + " was not found in the set.");
            }
        }
    }
}
```

OUTPUT

```
1 was not found in the
set.
2 was not found in the
set.
3 was found in the set.
4 was not found in the
set.
5 was not found in the
set.
6 was not found in the
set.
7 was found in the set.
8 was found in the set.
9 was not found in the
set.
10 was not found in
the set.
```

3. HashSet Methods

SN	Method	Description
1	**add(element)**	It is used to add the specified element to this set if it is not already present.
2	**clear()**	It is used to remove all of the elements from the set.
3	**contains(Object o)**	It is used to return true if this set contains the specified element.
4	**isEmpty()**	It is used to return true if this set contains no elements.
5	**iterator()**	It is used to return an iterator over the elements in this set.
6	**remove(element)**	It is used to remove the specified element from this set if it is present.
7	**size()**	It is used to return the number of elements in the set.

4. TreeSet class

- **TreeSet** class **implements** the **Set interface** that **uses a tree for storage.**
- It **stores the elements** in **ascending order.**
- It **uses a Tree structure** to **store elements.**
- It **contains unique elements** only **like HashSet.**
- It's **access** and **retrieval times** are **quite fast.**
- **How to create a LinkedList**

Syntax

```
TreeSet<Integer> numbers = new TreeSet<>();
```

- It **creates an empty tree** set that will be sorted in an **ascending order** according to the **natural order of the tree set**
- **TreeSet(Collection C)** //It creates a new tree set that contains the elements of the Collection C

TreeSet

```
//Program to Demonstrate TreeSet
import java.util.*;
class Demo
{
    public static void main(String
args[])
    {
        TreeSet t=new TreeSet();
        t.add(“Z");
        t.add(“D");
        t.add(“T");
        t.add(“a");
        System.out.println(t);
    }
}
```

OUTPUT

```
[D, T, Z,
a]
```

```
//Program to Demonstrate TreeSet
import java.util.*;
class TestTL2
{
    public static void main(String args[])
    {
        TreeSet t = new TreeSet();
        t.add("Akhil");
        t.add("Hemanth");
        t.add("Shiva");
        System.out.println("Ascending:");
        Iterator i=t.iterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }
        System.out.println("Descending:");
        Iterator j=t.descendingIterator();
        while(j.hasNext())
        {
            System.out.println(j.next());
        }
    }
}
```

OUTPUT

```
Ascending:
Akhil
Hemanth
Shiva
Descending:
Shiva
Hemanth
Akhil
```

4. TreeSet Methods

SNO	Method	Description
1	ceiling(E e)	It **returns the equal** or **closest greatest element** of the specified element from the set, or null there is no such element.
2	floor(E e)	It **returns the equal** or **closest least element** of the specified element from the set, or null there is no such element.
3	SortedSet headSet(Element)	It **returns the group of elements** that are **less than the specified element.**
4	higher(E e)	It returns the **closest greatest element** of the **specified element from the set**, or null there is no such element.
5	Iterator iterator()	It is used to **iterate the elements** in **ascending order.**
6	lower(E e)	It returns the **closest least element** of the **specified element** from the set, or null there is no such element.
7	pollFirst()	It is used to **retrieve and remove the lowest(first) element**.
8	pollLast()	It is used **to retrieve and remove the highest(last) element.**
9	E first()	It **returns the first (lowest) element** currently in this sorted set.
10	E last()	It returns the last (highest) element currently in this sorted set.

Delegation Event Model

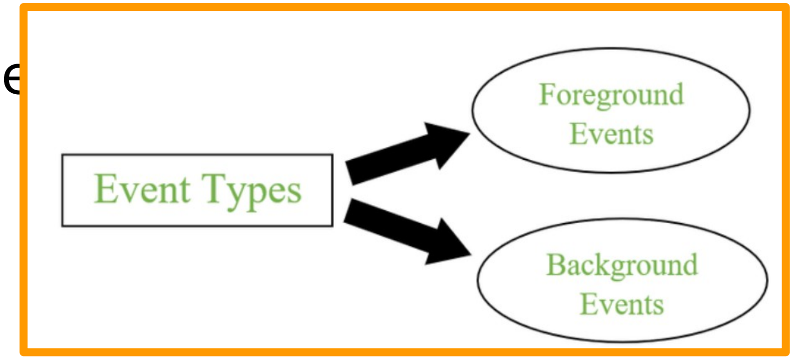
- The **Delegation Event Model** is a **programming pattern used** in Java for **handling events** in **graphical user interfaces (GUIs).**
- **Any program** that **uses GUI** is **event driven.**
- **Event** describes the **change in state** of **any object.**

Example : Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

- The **modern approach to event processing** is **based** on the **Delegation Model.**
- It defines a **standardized** and **compatible mechanism for generating** and **processing events.**
- In this approach, a **source generates** an **event** and **sends** it to **one** or **more listeners.**
- The **listener sits** and **waits** for an **event to occur.** When it gets an event, it is processed by the listener and returned.

Types of Events

- The events can be broadly classified into two cate



i. Foreground Events :

✓ Those events which require the direct interaction of user.

✓ They are generated as consequences of a person interacting with the graphical components in GUI.

Example:

✓ Clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

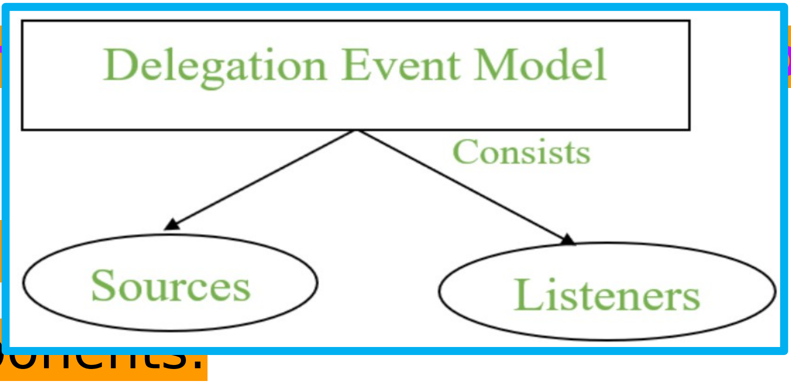
ii. Background Events:

✓ Events that don’t require interactions of users are known as background events.

Example :

What is Event Handling?

- It is a mechanism to control the events and to decide what should happen after an event occur.
- **Event handling** in Java is the **procedure that controls an event** and **performs appropriate action if it occurs**.
- **Event handlers** are **responsible** for **defining the actions** or behaviors **that should occur in response to specific events.**
- They contain the **implementation code** that **handles the event** and **performs** the desired **tasks.** Event handlers are typically implemented as methods within a class.
- **When an event occurs**, the associated **event h** **ond to that event.**
- To handle the events, Java follows the Delegation
- The Delegation Event model consists of two components.



Delegation Event Model

Consists

Sources

Listeners

Delegation Event model

1. Event Sources

- ✓ Event sources are **objects that generate events.**
- ✓ They are the entities or **components that trigger events** when **specific actions** or **conditions occur.**
- ✓ **Examples** of **event sources** include **buttons, text fields, mouse clicks**, or **keyboard inputs.**
- ✓ Event sources are **responsible for creating** and **dispatching** the corresponding **event objects** when the **specific event occurs.**

2. Event Listeners

- ✓ Event **listeners are interfaces** or classes **that define the methods** to **handle events.**
- ✓ They are **responsible for listening to events generated by event sources** and **invoking the appropriate event handlers** to respond to those events.

Registering the Source With Listene

- Different Classes provide different registration methods.

Syntax

addTypeListener()

- where Type represents the type of event.

Example 1:

- For KeyEvent we use addKeyListener() to register.

Example 2:

For ActionEvent we use addActionListener() to register.

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	Represents an action, such as a button click, triggered by a GUI component.
MouseEvent	MouseListener	Represents mouse events like clicks, enters, exits, and button presses on a GUI component.
KeyEvent	KeyListener	Represents keyboard events, such as key presses and releases, from a GUI component.
WindowEvent	WindowListener	Represents window-related events, like opening, closing, or resizing a GUI window.
FocusEvent	FocusListener	Represents focus-related events, including gaining and losing focus on a GUI component.

Java Event Classes and Listener Interfaces

i) Event Classes

- ActionEvent: This represents the user's action, such as clicking a button or selecting a menu item.
- MouseEvent: Represents mouse-related events, such as mouse clicks, movement, or dragging.
- KeyEvent: Represents keyboard-related events, such as key presses or key releases.
- WindowEvent: Represents events related to windows or frames, such as window opening, closing, or resizing.
- FocusEvent: Represents events related to focus, such as when a component gains or loses focus.

ii) Listener Interfaces

- ActionListener: Defines methods to handle ActionEvents.
- MouseListener: Defines methods to handle MouseEvent.
- MouseMotionListener: Defines methods to handle mouse motion events.
- KeyListener: Defines methods to handle KeyEvent.

Different interfaces consists of different methods

Listener Interface	Methods
ActionListener	•actionPerformed()
ComponentListener	•componentResized() •componentShown() •componentMoved() •componentHidden()
ItemListener	•itemStateChanged()
KeyListener	•keyTyped() •keyPressed() •keyReleased()
MouseListener	•mousePressed() •mouseClicked() •mouseEntered() •mouseExited() •mouseReleased()
MouseMotionListener	•mouseMoved() •mouseDragged()
MouseWheelListener	•mouseWheelMoved()
TextListener	•textChanged()

Flow of Event Handling

Step-1:

- ✓ **User Interaction** with a **component is required** to **generate an event.**

Step-2:

- ✓ The **object of the respective event class is created automatically** after event generation, and **it holds all information** of the **event source.**

Step-3:

- ✓ The **newly created object is passed** to the **methods of the registered listener.**

Step-4:

- ✓ The **method executes** and **returns the result.**

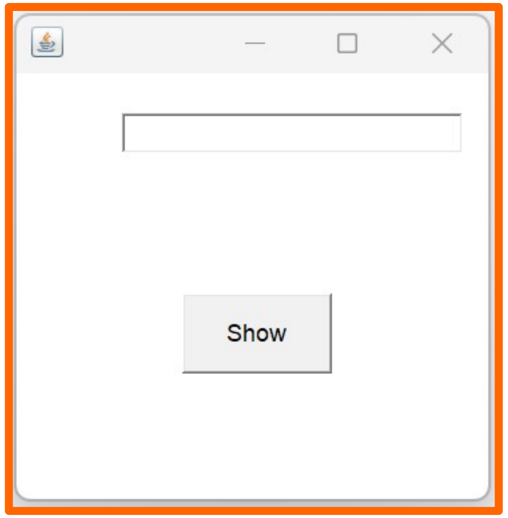
Simple Event Handling Program

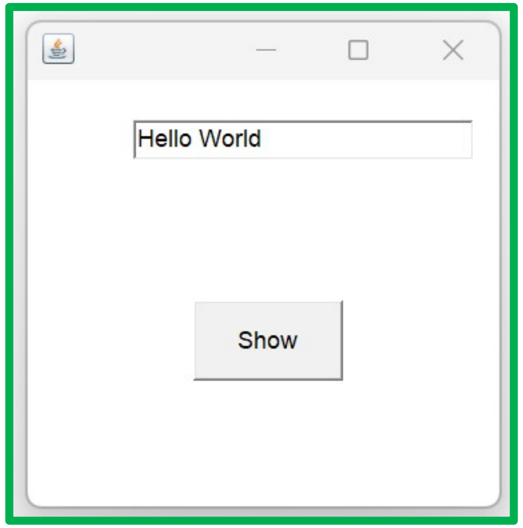
```
import java.awt.*;
import java.awt.event.*;

class EventHandling extends Frame implements
ActionListener
{
    EventHandling ()
    {
      TextField tf = new TextField ();
             tf.setBounds (60, 50, 170, 20);
            Button b = new Button ("Show");
            b.setBounds (90, 140, 75, 40);
            b.addActionListener (this);
            add (b);
            add (tf);
            setSize (250, 250);
             setLayout (null);
             setVisible (true);
    }
```

```
public void actionPerformed
(ActionEvent e)
    {
        tf.setText ("Hello World");
    }

    public static void main (String args[])
    {
         EventHandling eh=new
EventHandling ();
    }
}
```



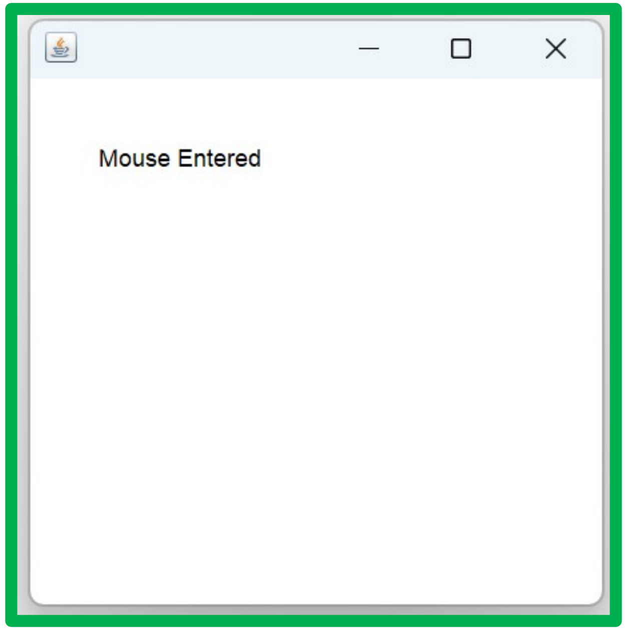

Hello World

Mouse Event handling Program

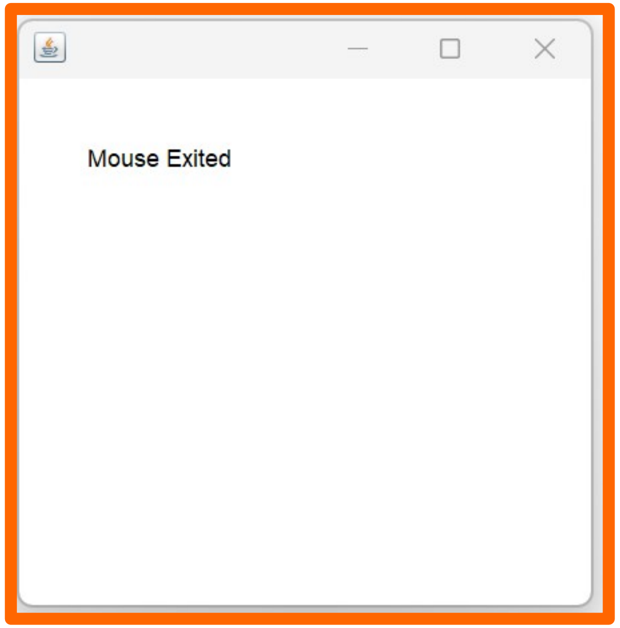
```
/* Java Program to demonstrate the event actions associated with
a mouse */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ML extends Frame implements MouseListener
{
    Label L;
    ML()
    {
        addMouseListener(this);
        L=new Label();
        L.setBounds(40,50,100,40);
        add(L);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseEntered(MouseEvent e)
    {
            L.setText("Mouse Entered");
    }
```

Mouse Event handling Program

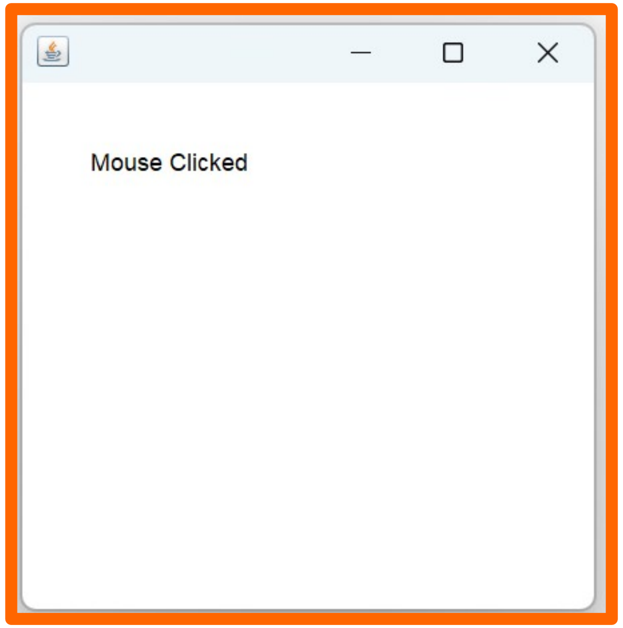
```
    public void mouseExited(MouseEvent
e)
    {
            L.setText("Mouse Exited");
    }
    public void
mouseReleased(MouseEvent e)
    {
            L.setText("Mouse Released");
    }
    public void mousePressed(MouseEvent
e)
    {
            L.setText("Mouse Pressed");
    }
    public void mouseClicked(MouseEvent
e)
    {
            L.setText("Mouse Clicked");
    }
    public static void main(String[] args)
    {
        ML obj=new ML();
    }
```



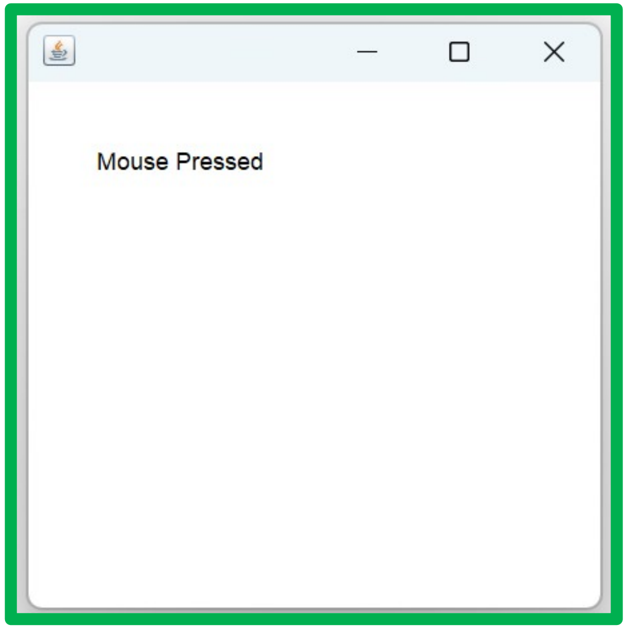
```
Mouse Entered
```



```
Mouse Exited
```



```
Mouse Clicked
```



```
Mouse Pressed
```

Key Event handling Program

```
import java.awt.*;
import java.awt.event.*;
class KeyDemo extends Frame implements
KeyListener
{
      String msg = "";
      String msg1= "";
      int X = 140, Y = 180;
      KeyDemo(String name)
       {
            super(name);
            setForeground(Color.red);
            addKeyListener(this);
      }
```

```
public void keyPressed(KeyEvent ke)
 {
 msg1= "Key Down";
 int key = ke.getKeyCode();
 switch(key)
 {
 case KeyEvent.VK_F1 : msg += "<F1>";
break;
 case KeyEvent.VK_F2 : msg += "<F2>";
break;
 case KeyEvent.VK_F3 : msg += "<F3>";
break;
 case KeyEvent.VK_PAGE_DOWN:msg +=
"<PgDn>";break;
 case KeyEvent.VK_PAGE_UP:msg +=
"<PgUp>";break;
 case KeyEvent.VK_LEFT:msg += "<Left
Arrow>";break;
 case KeyEvent.VK_RIGHT:msg += "<Right
Arrow>";break;
```

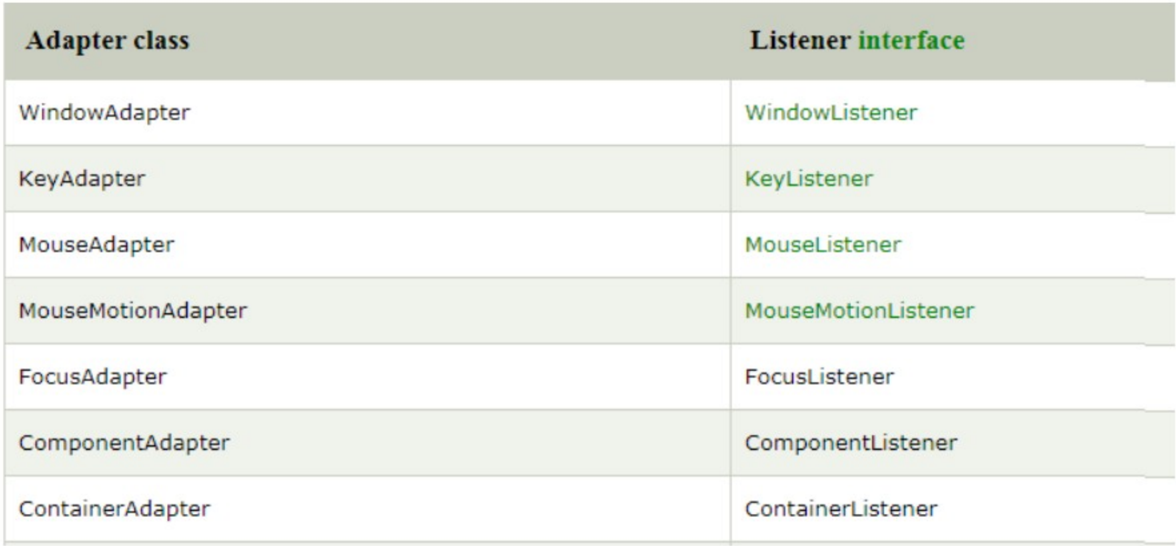
Key Event handling Program

```
public void keyReleased(KeyEvent ke)
{
     msg1="Key released";
     repaint();
}
public void keyTyped(KeyEvent ke)
{
     msg += ke.getKeyChar(); //gets the char st
     repaint();
}
 public void paint(Graphics g)
{
     Color c1 = new Color(123,50,89); //0 TO 255
rED,GREEN, BLUE
     g.setColor(c1);
     g.drawString(msg, X, Y);
     g.drawString(msg1,100,200);
}
 }
```

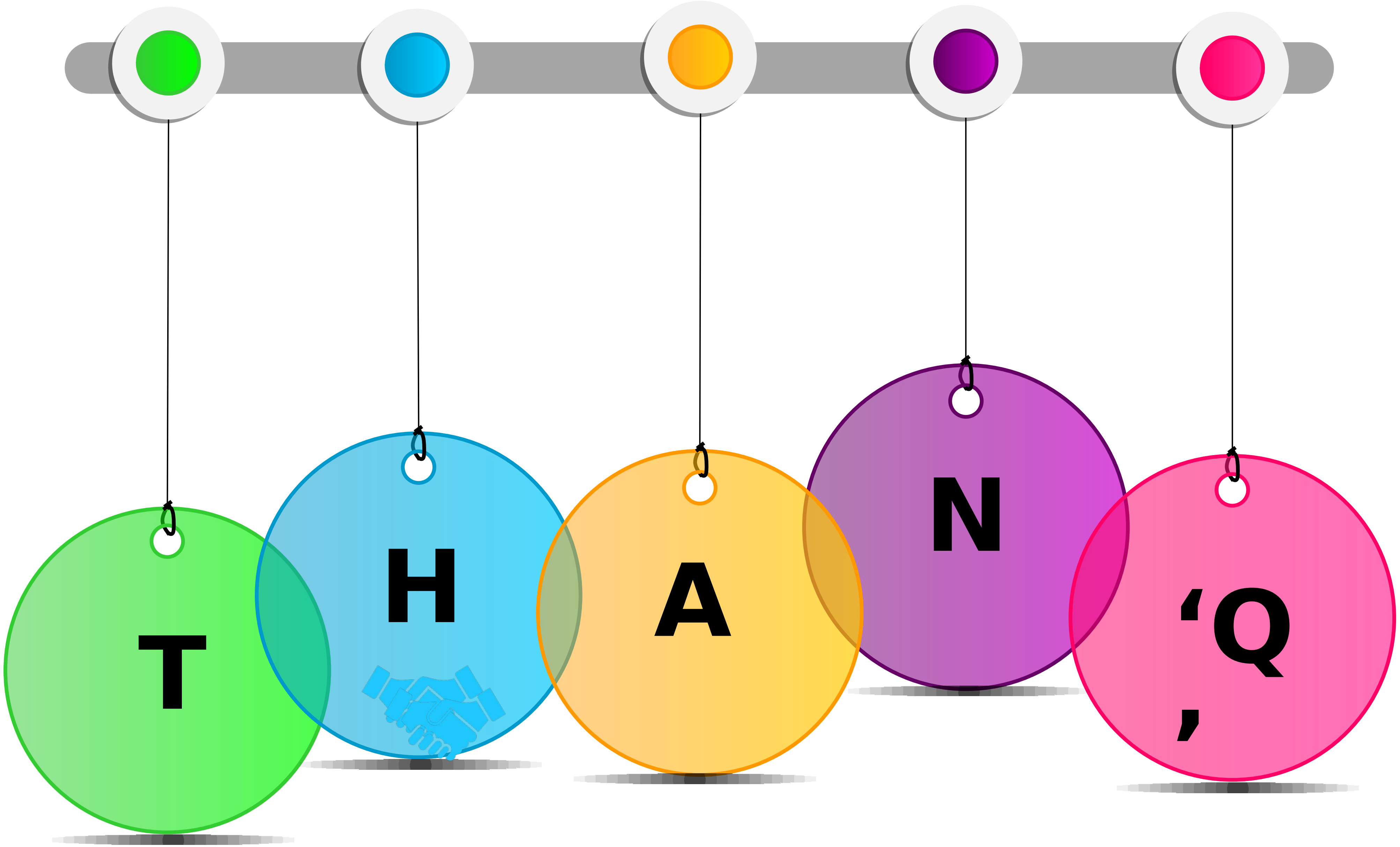
```
public class KeyDemo1
 {
public static void main(String args[])
 {
     KeyDemo f = new KeyDemo("Key
Events");
     f.setSize(300,400);
     f.setVisible(true);
 }
 }
```

Adapter Classes

- Java adapter classes provide the default implementation of listener interfaces.
- If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces.
- So it saves code.
- The adapter classes are found in java.awt.event and javax.swing.event packages.
- The Adapter classes with their corresponding listener interfaces are given below.

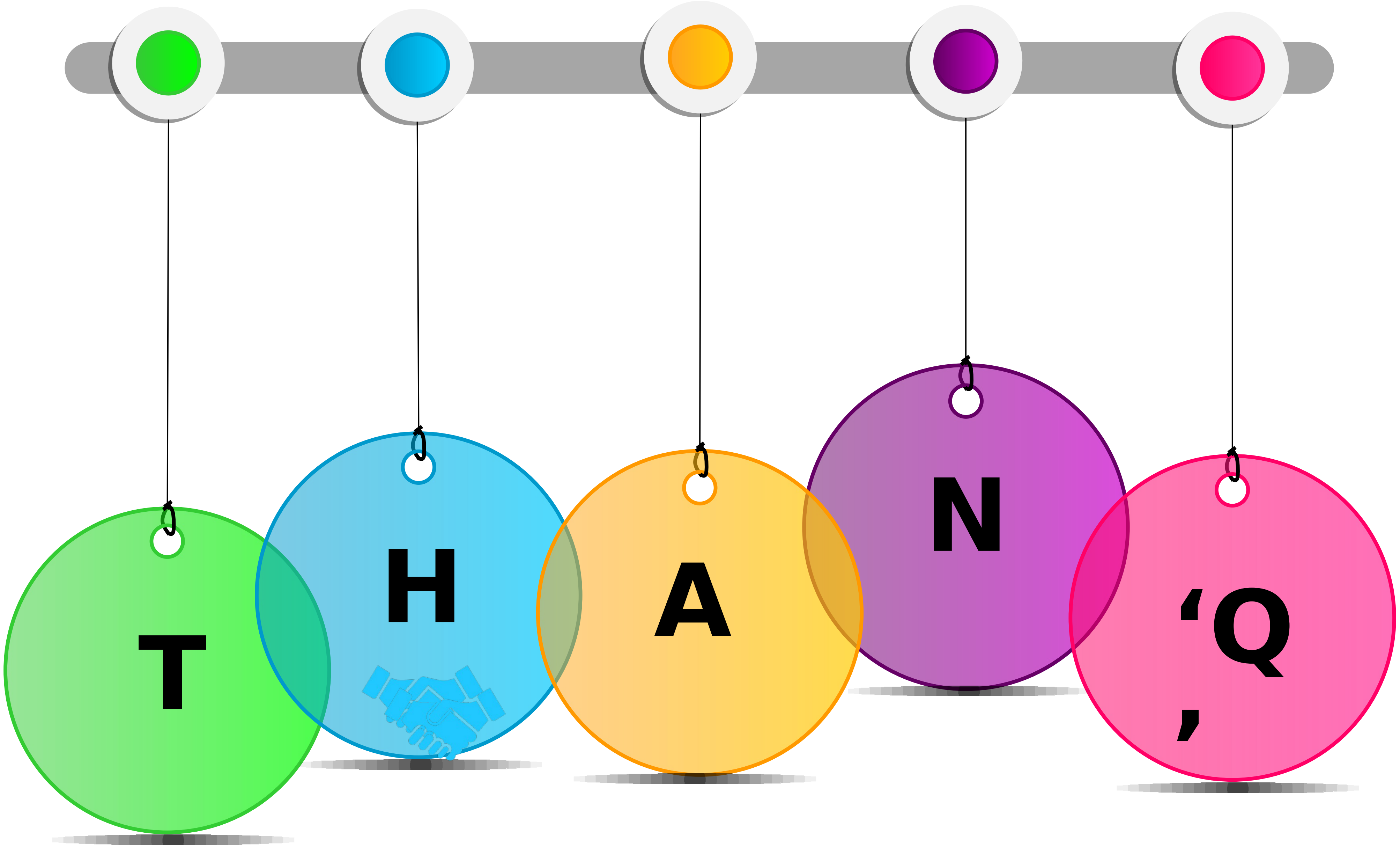


Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener



T H A N ‘Q

,



T

H

A

N

‘Q

;